stcal

Release 1.7.1.dev2+gfa14c81.d20240423

STScI <help@stsci.edu>

Apr 23, 2024

CONTENTS:

CHAPTER

ONE

API

1.1 stcal API

1.1.1 stcal Package

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

2.1 Package Documentation

2.1.1 Package Index

Jump Detection

Algorithm

This routine detects jumps by looking for outliers in the up-the-ramp signal for each pixel in each integration within an input exposure. On output, the GROUPDQ array is updated with the DQ flag "JUMP_DET" to indicate the location of each jump that was found. In addition, any pixels that have non-positive or NaN values in the gain reference file will have DQ flags "NO_GAIN_VALUE" and "DO_NOT_USE" set in the output PIXELDQ array. The SCI and ERR arrays of the input data are not modified.

The current implementation uses the two-point difference method described in Anderson & Gordon (2011).

Two-Point Difference Method

The two-point difference method is applied to each integration as follows:

1. Compute the first differences for each pixel (the difference between adjacent groups)
2. Compute the clipped median (dropping the largest difference) of the first differences for each pixel. If there are only three first difference values (four groups), no clipping is performed when computing the median.
3. Use the median to estimate the Poisson noise for each group and combine it with the read noise to arrive at an estimate of the total expected noise for each difference.
4. Compute the "difference ratio" as the difference between the first differences of each group and the median, divided by the expected noise.
5. If the largest "difference ratio" is greater than the rejection threshold, flag the group corresponding to that ratio as having a jump.
6. If a jump is found in a given pixel, iterate the above steps with the jump-impacted group excluded, looking for additional lower-level jumps that still exceed the rejection threshold.

7. Stop iterating on a given pixel when no new jumps are found or only one difference remains.
8. If there are only two differences (three groups), the smallest one is compared to the larger one and if the larger one is above a threshold, it is flagged as a jump.
9. If flagging of the 4 neighbors is requested, then the 4 adjacent pixels will have ramp jumps flagged in the same group as the central pixel as long as it has a jump between the min and max requested levels for this option.
10. If flagging of groups after a ramp jump is requested, then the groups in the requested time since a detected ramp jump will be flagged as ramp jumps if the ramp jump is above the requested threshold. Two thresholds and times are possible for this option.

Note that any ramp groups flagged as SATURATED in the input GROUPDQ array are not used in any of the above calculations and hence will never be marked as containing a jump.

Ramp Fitting

Description

This step determines the mean count rate, in units of counts per second, for each pixel by performing a linear fit to the data in the input file. The fit is done using the "ordinary least squares" method. The fit is performed independently for each pixel.

The count rate for each pixel is determined by a linear fit to the cosmic-ray-free and saturation-free ramp intervals for each pixel; hereafter this interval will be referred to as a "segment." The fitting algorithm uses an 'optimal' weighting scheme, as described by Fixsen et al. (2011).

Segments are determined using the 4-D GROUPDQ array of the input data set, under the assumption that the jump step will have already flagged CR's. Segments are terminated where saturation flags are found. Pixels are processed simultaneously in blocks using the array-based functionality of numpy. The size of the block depends on the image size and the number of groups.

Output Products

There are two output products created by default, with a third optional product also available:

1. The primary output file ("rate") contains slope and variance/error estimates for each pixel that are the result of averaging over all integrations in the exposure. This is a product with 2-D data arrays.
2. The secondary product ("rateints") contains slope and variance/error estimates for each pixel on a per-integration basis, stored as 3-D data cubes.
3. The third, optional, output product contains detailed fit information for every ramp segment for each pixel.

RATE Product

After computing the slopes and variances for all segments for a given pixel, the final slope is determined as a weighted average from all segments in all integrations, and is written to the "rate" output product. In this output product, the 4-D GROUPDQ from all integrations is collapsed into 2-D, merged (using a bitwise OR) with the input 2-D PIXELDQ, and stored as a 2-D DQ array. The 3-D VAR_POISSON and VAR_RNOISE arrays from all integrations are averaged into corresponding 2-D output arrays. In cases where the median rate for a pixel is negative, the VAR_POISSON is set to zero, in order to avoid the unphysical situation of having a negative variance.

RATEINTS Product

The slope images for each integration are stored as a data cube in "rateints" output data product. Each plane of the 3-D SCI, ERR, DQ, VAR_POISSON, and VAR_RNOISE arrays in this product corresponds to the result for a given integration. In this output product, the GROUPDQ data for a given integration is collapsed into 2-D and then merged with the input 2-D PIXELDQ array to create the output DQ array for each integration. The 3-D VAR_POISSON and VAR_RNOISE arrays are calculated by averaging over the fit segments in the corresponding 4-D variance arrays.

FITOPT Product

A third, optional output product is also available and is produced only when the step parameter `save_opt` is True (the default is False). This optional product contains 4-D arrays called SLOPE, SIGSLOPE, YINT, SIGYINT, WEIGHTS, VAR_POISSON, and VAR_RNOISE, which contain the slopes, uncertainties in the slopes, y-intercept, uncertainty in the y-intercept, fitting weights, variance of the slope due to poisson noise, and the variance of the slope due to read noise for each segment of each pixel, respectively. The y-intercept refers to the result of the fit at an effective exposure time of zero. This product also contains a 3-D array called PEDESTAL, which gives the signal at zero exposure time for each pixel, and the 4-D CRMAG array, which contains the magnitude of each group that was flagged as having a CR hit.

By default, the name of this output file will have the product type suffix "_fitopt". In this optional output product, the pedestal array is calculated for each integration by extrapolating the final slope (the weighted average of the slopes of all ramp segments in the integration) for each pixel from its value at the first group to an exposure time of zero. Any pixel that is saturated on the first group is given a pedestal value of 0. Before compression, the cosmic-ray magnitude array is equivalent to the input SCI array but with the only nonzero values being those whose pixel locations are flagged in the input GROUPDQ as cosmic ray hits. The array is compressed, removing all groups in which all the values are 0 for pixels having at least one group with a non-zero magnitude. The order of the cosmic rays within the ramp is preserved.

Special Cases

If the input dataset has only one group in each integration (NGROUPS=1), the count rate for all unsaturated pixels in each integration will be calculated as the value of the science data in the one group divided by the group time. If the input dataset has only two groups per integration (NGROUPS=2), the count rate for all unsaturated pixels in each integration will be calculated using the differences between the two valid groups of the science data divided by the group time.

For datasets having more than one group in each integration (NGROUPS>1), a ramp having a segment with only one good group is processed differently depending on the number and size of the other segments in the ramp. If a ramp has only one segment and that segment contains a single group, the count rate will be calculated to be the value of the science data in that group divided by the group time. If a ramp has a segment with only one good group, and at least one other segment having more than one good group, only data from the segment(s) having more than one good group will be used to calculate the count rate.

For ramps in a given integration that are saturated beginning in their second group, the count rate for that integration will be calculated as the value of the science data in the first group divided by the group time, but only if the step parameter `suppress_one_group` is set to `False`. If set to `True`, the computation of slopes for pixels that have only one good group will be suppressed and the slope for that integration will be set to zero.

Slope and Variance Calculations

Slopes and their variances are calculated for each segment, for each integration, and for the entire exposure. As defined above, a segment is a set of contiguous groups where none of the groups is saturated or cosmic ray-affected. The appropriate slopes and variances are output to the primary output product, the integration-specific output product, and the optional output product. The following is a description of these computations. The notation in the equations is the following: the type of noise (when appropriate) will appear as the superscript 'R', 'P', or 'C' for readnoise, Poisson noise, or combined, respectively; and the form of the data will appear as the subscript: 's', 'i', 'o' for segment, integration, or overall (for the entire dataset), respectively.

It is possible for an integration or pixel to have invalid data, so usable slope data will not be available. If a pixel has an invalid integration, the value for that integration for that pixel will be set to NaN in the rateints product. Further, if all integrations for a given pixel are invalid the pixel value for the rate product will be set to NaN. An example of invalid data would be a fully saturated integration for a pixel.

Optimal Weighting Algorithm

The slope of each segment is calculated using the least-squares method with optimal weighting, as described by Fixsen et al 2000 and Regan 2007, JWST-STScI-001212. Optimal weighting determines the relative weighting of each sample when calculating the least-squares fit to the ramp. When the data have low signal-to-noise ratio S, the data are read noise dominated and equal weighting of samples is the best approach. In the high signal-to-noise regime, data are Poisson-noise dominated and the least-squares fit is calculated with the first and last samples. In most practical cases, the data will fall somewhere in between, where the weighting is scaled between the two extremes.

The signal-to-noise ratio S used for weighting selection is calculated from the last sample as:

$$S = \frac{data \times gain}{\sqrt{(read\_noise)^2 + (data \times gain)}},$$

The weighting for a sample i is given as:

$$w_i = (i - i_{midpoint})^P,$$

where $i_{midpoint}$ is the the sample number of the midpoint of the sequence, and P is the exponent applied to weights, determined by the value of S. Fixsen et al. 2000 found that defining a small number of P values to apply to values of S was sufficient; they are given as:

Minimum S	Maximum S	P
0	5	0
5	10	0.4
10	20	1
20	50	3
50	100	6
100		10

Segment-specific Computations

The variance of the slope of a segment due to read noise is:

$$var_s^R = \frac{12\ R^2}{(ngroups_s^3 - ngroups_s)(tgroup^2)}\,,$$

where R is the noise in the difference between 2 frames, $ngroups_s$ is the number of groups in the segment, and $tgroup$ is the group time in seconds (from the keyword TGROUP).

The variance of the slope in a segment due to Poisson noise is:

$$var_s^P = \frac{slope_{est} + darkcurrent}{tgroup \times gain\ (ngroups_s - 1)}\,,$$

where $gain$ is the gain for the pixel (from the GAIN reference file), in e/DN. The $slope_{est}$ is an overall estimated slope of the pixel, calculated by taking the median of the first differences of the groups that are unaffected by saturation and cosmic rays, in all integrations. This is a more robust estimate of the slope than the segment-specific slope, which may be noisy for short segments. The contributions from the dark current are added when present; the value can be provided by the user during the *jwst.dark_current.DarkCurrentStep*, or it can be specified in scalar or 2D array form by the dark reference file.

The combined variance of the slope of a segment is the sum of the variances:

$$var_s^C = var_s^R + var_s^P$$

Integration-specific computations

The variance of the slope for an integration due to read noise is:

$$var_i^R = \frac{1}{\sum_s \frac{1}{var_s^R}}\,,$$

where the sum is over all segments in the integration.

The variance of the slope for an integration due to Poisson noise is:

$$var_i^P = \frac{1}{\sum_s \frac{1}{var_s^P}}$$

The combined variance of the slope for an integration due to both Poisson and read noise is:

$$var_i^C = \frac{1}{\sum_s \frac{1}{var_s^R + var_s^P}}$$

The slope for an integration depends on the slope and the combined variance of each segment's slope:

$$slope_i = \frac{\sum_s \frac{slope_s}{var_s^C}}{\sum_s \frac{1}{var_s^C}}$$

Exposure-level computations

The variance of the slope due to read noise depends on a sum over all integrations:

$$var_o^R = \frac{1}{\sum_i \frac{1}{var_i^R}}$$

The variance of the slope due to Poisson noise is:

$$var_o^P = \frac{1}{\sum_i \frac{1}{var_i^P}}$$

The combined variance of the slope is the sum of the variances:

$$var_o^C = var_o^R + var_o^P$$

The square-root of the combined variance is stored in the ERR array of the output product.

The overall slope depends on the slope and the combined variance of the slope of each integration's segments, and hence is a sum over integrations and segments:

$$slope_o = \frac{\sum_{i,s} \frac{slope_{i,s}}{var_{i,s}^C}}{\sum_{i,s} \frac{1}{var_{i,s}^C}}$$

Error Propagation

Error propagation in the `ramp_fitting` step is implemented by carrying along the individual variances in the slope due to Poisson noise and read noise at all levels of calculations. The total error estimate at each level is computed as the square-root of the sum of the two variance estimates.

In each type of output product generated by the step, the variance in the slope due to Poisson noise is stored in the "VAR_POISSON" extension, the variance in the slope due to read noise is stored in the "VAR_RNOISE" extension, and the total error is stored in the "ERR" extension. In the optional output product, these arrays contain information for every segment used in the fitting for each pixel. In the "rateints" product they contain values for each integration, and in the "rate" product they contain values for the exposure as a whole.

Data Quality Propagation

For a given pixel, if all groups in an integration are flagged as DO_NOT_USE or SATURATED, then that pixel will be flagged as DO_NOT_USE in the corresponding integration in the "rateints" product. Note this does NOT mean that all groups are flagged as SATURATED, nor that all groups are flagged as DO_NOT_USE. For example, slope calculations that are suppressed due to a ramp containing only one good group will be flagged as DO_NOT_USE in the first group, but not necessarily any other group, while only groups two and beyond are flagged as SATURATED. Further, only if all integrations in the "rateints" product are flagged as DO_NOT_USE, then the pixel will be flagged as DO_NOT_USE in the "rate" product.

For a given pixel, if all groups in an integration are flagged as SATURATED, then that pixel will be flagged as SATURATED and DO_NOT_USE in the corresponding integration in the "rateints" product. This is different from the above case in that this is only for all groups flagged as SATURATED, not for some combination of DO_NOT_USE and SATURATED. Further, only if all integrations in the "rateints" product are flagged as SATURATED, then the pixel will be flagged as SATURATED and DO_NOT_USE in the "rate" product.

For a given pixel, if any group in an integration is flagged as JUMP_DET, then that pixel will be flagged as JUMP_DET in the corresponding integration in the "rateints" product. That pixel will also be flagged as JUMP_DET in the "rate" product.

Alignment Utils

Description

This sub-package contains all the modules common to all missions.

stcal.alignment Package

Functions

calc_rotation_matrix(roll_ref, v3i_yangle[, ...])	Calculate the rotation matrix.
compute_fiducial(wcslist[, bounding_box])	Calculates the world coordinates of the fiducial point of a list of WCS objects.
compute_scale(wcs, fiducial[, disp_axis, ...])	Compute the scale at the fiducial point on the detector..
reproject(wcs1, wcs2)	Given two WCSs or transforms return a function which takes pixel coordinates in the first WCS or transform and computes them in pixel coordinates in the second one.
wcs_from_footprints(dmodels[, refmodel, ...])	Create a WCS from a list of input datamodels.

calc_rotation_matrix

stcal.alignment.**calc_rotation_matrix**(*roll_ref: float*, *v3i_yangle: float*, *vparity: int = 1*) → list[float]

Calculate the rotation matrix.

Parameters

- **roll_ref** (*float*) – Telescope roll angle of V3 North over East at the ref. point in radians
- **v3i_yangle** (*float*) – The angle between ideal Y-axis and V3 in radians.
- **vparity** (*int*) – The x-axis parity, usually taken from the JWST SIAF parameter VIdlParity. Value should be "1" or "-1".

Returns

matrix – A list containing the rotation matrix elements in column order.

Return type

list

Notes

The rotation matrix is

$$PC = beginbmatrix pc_{1,1} pc_{2,1}$$

$$pc_{1,2} pc_{2,2} endbmatrix$$

compute_fiducial

stcal.alignment.**compute_fiducial**(*wcslist: list*, *bounding_box: tuple | list | None = None*) → ndarray

Calculates the world coordinates of the fiducial point of a list of WCS objects. For a celestial footprint this is the center. For a spectral footprint, it is the beginning of its range.

Parameters

- **wcslist** (*list*) – A list containing all the WCS objects for which the fiducial is to be calculated.
- **bounding_box** (*tuple, list, None*) – The bounding box over which the WCS is valid. It can be a either tuple of tuples or a list of lists of size 2 where each element represents a range of (low, high) values. The bounding_box is in the order of the axes, axes_order. For two inputs and axes_order(0, 1) the bounding box can be either ((xlow, xhigh), (ylow, yhigh)) or [[xlow, xhigh], [ylow, yhigh]].

Returns

fiducial – A two-elements array containing the world coordinates of the fiducial point in the combined output coordinate frame.

Return type

numpy.ndarray

Notes

This function assumes all WCSs have the same output coordinate frame.

compute_scale

stcal.alignment.**compute_scale**(*wcs: WCS*, *fiducial: tuple | ndarray*, *disp_axis: int | None = None*, *pscale_ratio: float | None = None*) → float

Compute the scale at the fiducial point on the detector..

Parameters

- **wcs** (*WCS*) – Reference WCS object from which to compute a scaling factor.
- **fiducial** (*tuple*) – Input fiducial of (RA, DEC) or (RA, DEC, Wavelength) used in calculating reference points.
- **disp_axis** (*int*) – Dispersion axis integer. Assumes the same convention as `wcsinfo.dispersion_direction`
- **pscale_ratio** (*int*) – Ratio of input to output pixel scale

Returns

scale – Scaling factor for x and y or cross-dispersion direction.

Return type

float

reproject

stcal.alignment.**reproject**(*wcs1*, *wcs2*)

Given two WCSs or transforms return a function which takes pixel coordinates in the first WCS or transform and computes them in pixel coordinates in the second one. It performs the forward transformation of `wcs1` followed by the inverse of `wcs2`.

Parameters

- **wcs1** (*astropy.wcs.WCS or gwcs.wcs.WCS*) – Input WCS objects or transforms.
- **wcs2** (*astropy.wcs.WCS or gwcs.wcs.WCS*) – Output WCS objects or transforms.

Returns

- *Function to compute the transformations. It takes x, y*
- positions in `wcs1` and returns x, y positions in `wcs2`.

wcs_from_footprints

stcal.alignment.**wcs_from_footprints**(*dmodels*, *refmodel=None*, *transform=None*, *bounding_box=None*, *pscale_ratio=None*, *pscale=None*, *rotation=None*, *shape=None*, *crpix=None*, *crval=None*)

Create a WCS from a list of input datamodels.

A fiducial point in the output coordinate frame is created from the footprints of all WCS objects. For a spatial frame this is the center of the union of the footprints. For a spectral frame the fiducial is in the beginning of the footprint range. If `refmodel` is None, the first WCS object in the list is considered a reference. The output coordinate frame and projection (for celestial frames) is taken from `refmodel`. If `transform` is not supplied, a compound transform is created using CDELTs and PC. If `bounding_box` is not supplied, the *bounding_box* of the new WCS is computed from *bounding_box* of all input WCSs.

Parameters

- **dmodels** (*list*) – A list of valid datamodels.
- **refmodel** – A valid datamodel whose WCS is used as reference for the creation of the output coordinate frame, projection, and scaling and rotation transforms. If not supplied the first model in the list is used as `refmodel`.
- **transform** (*Model*) – A transform, passed to `gwcs.wcstools.wcs_from_fiducial()` If not supplied *Scaling | Rotation* is computed from `refmodel`.
- **bounding_box** (*tuple*) – Bounding_box of the new WCS. If not supplied it is computed from the bounding_box of all inputs.
- **pscale_ratio** (*float, None*) – Ratio of input to output pixel scale. Ignored when either `transform` or `pscale` are provided.
- **pscale** (*float, None*) – Absolute pixel scale in degrees. When provided, overrides `pscale_ratio`. Ignored when `transform` is provided.
- **rotation** (*float, None*) – Position angle of output image's Y-axis relative to North. A value of 0.0 would orient the final output image to be North up. The default of *None* specifies that the images will not be rotated, but will instead be resampled in the default orientation for the camera with the x and y axes of the resampled image corresponding approximately to the detector axes. Ignored when `transform` is provided.

- **shape** (*tuple of int, None*) – Shape of the image (data array) using `numpy.ndarray` convention (`ny` first and `nx` second). This value will be assigned to `pixel_shape` and `array_shape` properties of the returned WCS object.
- **crpix** (*tuple of float, None*) – Position of the reference pixel in the image array. If `crpix` is not specified, it will be set to the center of the bounding box of the returned WCS object.
- **crval** (*tuple of float, None*) – Right ascension and declination of the reference pixel. Automatically computed if not provided.

Returns

wcs_new – The WCS object corresponding to the combined input footprints.

Return type

WCS

PYTHON MODULE INDEX

S

INDEX